Formal Verification

Formal verification of software programs involves proving that a program satisfies a formal specification of its behavior. Subareas of formal verification include deductive verification (see above), abstract interpretation, automated theorem proving, type systems, and lightweight formal methods. A promising type-based verification approach is dependently typed programming, in which the types of functions include (at least part of) those functions' specifications, and type-checking the code establishes its correctness against those specifications. Fully featured dependently typed languages support deductive verification as a special case.

Another complementary approach is program derivation, in which efficient code is produced from functional specifications by a series of correctness-preserving steps. An example of this approach is the Bird-Meertens Formalism, and this approach can be seen as another form of correctness by construction.

These techniques can be sound, meaning that the verified properties can be logically deduced from the semantics, or unsound, meaning that there is no such guarantee. A sound technique yields a result only once it has searched the entire space of possibilities. An example of an unsound technique is one that searches only a subset of the possibilities, for instance only integers up to a certain number, and give a "good-enough" result. Techniques can also be decidable, meaning that their algorithmic implementations are guaranteed to terminate with an answer, or undecidable, meaning that they may never terminate. Because they are bounded, unsound techniques are often more likely to be decidable than sound ones.